

# PyCraters: A Python framework for crater function analysis

Scott A. Norris

October 31, 2014

## Abstract

We introduce a Python framework designed to automate the most common tasks associated with the extraction and upscaling of the statistics of single-impact crater functions to inform coefficients of continuum equations describing surface morphology evolution. Designed with ease-of-use in mind, the framework allows users to extract meaningful statistical estimates with very short Python programs. Wrappers to interface with specific simulation packages, routines for statistical extraction of output, and fitting and differentiation libraries are all hidden behind simple, high-level user-facing functions. In addition, the framework is extensible, allowing advanced users to specify the collection of specialized statistics or the creation of customized plots. The framework is hosted on the BitBucket service under an open-source license, with the aim of helping non-specialists easily extract preliminary estimates of relevant crater function results associated with a particular experimental system.

## 1 Introduction

Irradiation by energetic ions is a ubiquitous materials processing technique, used widely in laboratories and industry for doping, cleaning, and modification of materials surfaces. Under certain environmental conditions, ion irradiation is observed to induce the spontaneous formation of nanometer-scale structures such as ripples, dots, and holes [1]. In some contexts, such as the gradual ion-induced degradation of fusion reactor components, these structures are an artifact to be avoided [2], but more recent observations of well-ordered, high-aspect ratio structures [3] has led to the consideration of ion irradiation as an inexpensive means of inducing such structures deliberately. With sufficient understanding, this process could serve as the basis of an inexpensive, high-throughput means of creating surfaces with desired mechanical, optical, or electronic properties, ready for immediate application across the existing installed base of ion beam facilities.

A major barrier to predictive understanding of the ion-induced nanostructuring process has been a large number of competing physical mechanisms, and an accompanying difficulty in estimating their relative magnitudes in different parameter regimes. To the original modeling of energy deposition and its relationship to sputter yield by Sigmund [4, 5], and subsequent discovery of an sputter erosion-driven instability mechanism identified by Bradley and Harper [6], numerous additional mechanisms have since been added. It has since been discovered that atoms displaced during the collision cascade, but not sputtered from the surface, produce contributions to the evolution equations that directly compete with the erosive mechanism. First studied in 1D by Carter and Vishnyakov, and later in 2D dimensions by Davidovitch et al. [7, 8], this "momentum transfer" or "mass redistribution" contribution effectively doubles the number of unknown parameters in the problem, as every erosive term has a redistributive counterpart, and the magnitudes of each must in principle be estimated. Intensifying this problem was the realization that strongly-ordered structures generating the most excitement seemed to be due to the presence of multiple components in the target [9, 10], whether through contamination, intentional co-deposition, or the use of a two-component target such as III-V compounds. To accurately model such equations requires the introduction of a second field to track concentrations [11] which, although it can overcome the barriers to ordered structures exhibited by one-component models [12, 13], again doubles the number of unknown parameters present in the problem.

Recently, the "Crater Function" framework has emerged as a means of rigorously connecting surface morphology evolution over long spatial and temporal scales to the statistical properties of single ion impacts [14, 15, 16]. Given the "Crater Function"  $\Delta h(x, y; \mathcal{S})$  describing the average surface modification due to an ion impact (with a parametric dependence on the surface properties via the argument  $\mathcal{S}$ ), the multi-scale

analysis within the framework [14] produces contributions to partial differential equations governing the surface morphology evolution. It can therefore be viewed as a way of estimating many of the unknown parameters present in these problems by means of atomistic simulation. Originally applied only to pure materials and using only data from flat surfaces [15], the framework has since been expanded to the case of binary materials [16], and to enable incorporation of simulation data from curved targets [17]. It has thus matured to the point where it may be of value to the general community as a parameter estimation tool. However, to use the framework to this end, one needs the capability to (a) perform numerical simulations of ion impacts over a variety of surface parameters, (b) extract the necessary statistics from the output of each simulations, (c) fit parameter-dependent statistics to various appropriate functional forms, and (d) combine and report the results.

This manuscript describes a Python library to provide all of the above capabilities through a simple and user-friendly API. Access to various simulation tools is provided via wrappers that automatically create input files, run the solver, read the output, and save in a common format. A customizable set of statistical analyses are then run on the common-format output file, and saved for later use under a unique parameter-dependent filename. A flexible loading mechanism and general purpose fitting library make it easy to load statistics as a function of arbitrary parameters, and then fit the resulting data points to appropriate smooth nonlinear functions. Finally, functions utilizing these capabilities are provided to perform all current mathematical operations indicated by the literature to extract and plot PDE components. Using this library, example codes that re-obtain existing results within the literature are as short as 20 lines each, and can easily be modified by end users to begin studying systems of their choosing.

## 2 Theoretical Background

**Crater Functions.** If the dominant effects of the impact-induced collision cascade can be assumed to take place near to the surface of the evolving film, then the normal surface velocity of an ion-bombarded surface can be represented by an integro-differential equation of the form [18],

$$v_n = \int I(\phi(\mathbf{x}')) \Delta h(\mathbf{x} - \mathbf{x}'; \mathcal{S}(\mathbf{x}')) d\mathbf{x}', \quad (1)$$

where  $I = I_0 \cos(\phi)$  is the projected ion flux depending on the local angle of incidence  $\phi(\mathbf{x})$ ,  $\Delta h$  is the “crater function” describing the average surface response due to single ion impacts, and  $\mathcal{S}$  describes an arbitrary parametric dependence of the crater function on the local surface shape. This form has advantages over more traditional treatments of irradiation-induced morphology evolution. Instead of separate, simplified models of the processes of ion-induced sputtering [5, 6] and impact-induced momentum transfer [7, 8] – both of which break down as the angle of incidence approaches grazing – the crater function  $\Delta h$  naturally includes components due to both sputtered atoms and redistributed atoms (thus unifying the two approaches), and can in principle be obtained empirically (thus avoiding inaccuracy at high angles of incidence).

**A Generic Framework.** Exploiting the typical experimental observation of a separation of spatial scales between the size of the impact (direct spatial dependence of  $\Delta h$ ) and the typical size of emergent structures (spatial dependence of  $\phi$  and  $\mathcal{S}$ ), a multiple-scale analysis was conducted in which the integral in Eq. (1) is expanded into an infinite series of terms involving the moments of  $\Delta h$  [14]:

$$v_n = \left[ I \tilde{M}^{(0)} \right] + \varepsilon \nabla_S \cdot \left[ I \tilde{M}^{(1)} \right] + \frac{1}{2} \varepsilon^2 \nabla_S \cdot \nabla_S \cdot \left[ I \tilde{M}^{(2)} \right] + \dots \quad (2)$$

where  $I$  is the projected ion flux (a scalar), the  $\nabla_S$  are co-ordinate free surface divergences, and the  $\tilde{M}^{(i)}$  are “effective” moments of the crater function  $\Delta h$  in increasing tensor order [14]. This result provides intuition as to which parts of the crater function  $\Delta h$  are most important to understand morphology evolution, and represents a general solution for the multiple-scale expansion of Eq. (1) in the sense that it should apply for any parametric dependencies of the crater function on the surface shape  $\mathcal{S}$ . (Note that in this co-ordinate free form, the effective moments are really combinations of the actual moments as described in Ref. [14]; however, in any linearization, they are equivalent).

**Example Applications.** Equation (2) is fully nonlinear and independent of the specific form of the crater function. Therefore, to study surface stability, one must first choose a form for the crater function  $\Delta h$ , and then linearize the resulting specific instance of Equation (2) about a flat surface. This process was first demonstrated in Ref. [15], where for simplicity and consistency with available simulation data, the crater function

$$\Delta h = g(\mathbf{x} - \mathbf{x}'; \phi), \quad (3)$$

was chosen, which depends parametrically only on the local angle of incidence  $\phi$ . Inserting this expression into the general result Eq. (2), linearizing, and then adopting a moving frame of reference to eliminate translational and advective terms, one obtains to leading order the PDE

$$\frac{\partial h}{\partial t} = S_X(\theta) \frac{\partial^2 h}{\partial x^2} + S_Y(\theta) \frac{\partial^2 h}{\partial y^2} \quad (4)$$

where the angle-dependent coefficients  $S_X(\theta)$  and  $S_Y(\theta)$  are related to the crater functions via the expressions

$$\begin{aligned} S_X(\theta) &= \frac{\partial}{\partial \theta} \left[ I_0 \cos(\theta) M_x^{(1)}(\theta) \right] \\ S_Y(\theta) &= \left[ I_0 \cos(\theta) \cot(\theta) M_x^{(1)}(\theta) \right] \end{aligned} \quad (5)$$

where  $M_x^{(1)}$  is the component of the (vector) first moment in the projected direction of the ion beam. More recently, a similar approach has been applied to an extended crater function of the form [17]

$$\Delta h = g(\mathbf{x} - \mathbf{x}'; \phi, K_{11}, K_{12}, K_{22}) \quad (6)$$

depending additionally on the surface curvatures  $K_{ij}$  near the point of impact. It was found that including this dependency within the crater function reveals additional terms in the coefficient values, which take the revised form

$$\begin{aligned} S_X(\theta) &= \frac{\partial}{\partial \theta} \left[ I_0 \cos(\theta) M_x^{(1)}(\theta) \right] + \frac{\partial}{\partial K_{11}} \left[ I_0 \cos(\theta) M^{(0)} \right] \\ S_Y(\theta) &= \left[ I_0 \cos(\theta) \cot(\theta) M_x^{(1)}(\theta) \right] + \frac{\partial}{\partial K_{22}} \left[ I_0 \cos(\theta) M^{(0)} \right] \end{aligned} \quad (7)$$

**Implications.** The practical consequence of such results is that one can directly connect atomistic simulations over small length- and time-scales to continuum equations governing morphology evolution over much longer scales. If the crater function  $\Delta h$  and its moments  $M^{(i)}$  can be identified as functions of the local surface configuration (angle, curvature, etc.) by simulation (e.g. [19]) or even experiment (e.g. [20]), then the expected continuum evolution of the system can be predicted via Eq. (4) with coefficients supplied by results such as Eqs. (5) or (7). Early applications of these ideas show significant promise for predicting the outcome of experiments [15] or determining the likely physical mechanisms driving experimental observations [16]. However, the steps required to do so represent a non-trivial task in simulation and data analysis: an effective procedure must address questions of

- (1) creation or selection of a simulation tool to perform many single-impact simulations
- (2) obtaining statistically converged moments at individual parameter combinations,
- (3) estimation of derivative values using data from adjacent parameter combinations
- (4) a smoothing mechanism to prevent uncertainties at step (2) from being amplified in step (3)

An approach incorporating such steps was first demonstrated in Ref. [15], where molecular dynamics simulations using the PARCAS code [21] were performed for irradiation of Si by 250 eV Ar<sup>+</sup> at 5-degree increments between 0° and 90°. Smoothing was accomplished by a weighted fitting of the simulation results to a truncated Fourier series, and fitted values of  $M_x^{(1)}(\theta)$  were then inserted into Eqs. (5). Analyzing the resulting linear PDE of the form (4) (with additional terms describing ion-enhanced viscous flow), the most-unstable

wavelengths at each angle were compared to the wavelengths of experimentally-observe structures, with reasonable agreement. In the process, the relative sizes of the effects of erosion and redistribution were directly obtained and compared, and the effects of redistribution were unexpectedly found to be dominant for the chosen system.

### 3 Goals and Outline of the Framework

The previous section outlines the general features of the Crater Function approach, including the potential promise for the general problem of coefficient estimation, and also the technical hurdles associated with its use. However, while the process of simulation, statistical analysis, fitting, and differentiation is time-consuming, it is also in principle mechanical, suggesting the utility of an open-source library to centralize best practices and avoid repeated re-implementations. Here we describe the primary goals of such a library, and a summary of the structure of the resulting codebase.

**Motivation #1: Accessibility.** A principal motivation for the present work is to automate the process described above in a generic and accessible way. As many of the procedures as possible should be performed automatically, with reasonable default strategies applied for users that do not wish to delve into the details of atomistic simulation, internal statistical data structures, or the optimal fitting functions for moment curves. Instead, a first-time user should be able to spend most of their effort on deciding what system and environmental parameters to study, after which the library should take care of subsequent mechanical operations. In particular, it should provide simple visualization routines for tasks expected to be common, such as plotting the calculated angle-dependent coefficient values.

**Motivation #2: Extensibility.** A second motivation is to accommodate the desires of users who wish to move beyond the basic capabilities just described. For example, whereas first-time users may wish to call a high-level function to obtain a graph similar to one already published in the literature, a more advanced user might like to work directly with the raw statistical data, using the collected moments to extract particular quantities of interest. Finally, a very advanced user or researcher within the field may wish to customize the set of statistics to be gathered, or even modify the methods used to calculate and fit those customized statistical quantities. As much as possible, the goal should be to accommodate users at each of these levels of detail, and allowing each of them to use built-in utilities to simplify the calculation, loading, fitting, and plotting of whatever quantities are of interest to a given user.

**Motivation #3: Portability.** A final motivation is to enable the collection of statistical data from as many sources as possible, with as little effort as possible. When first demonstrated in Ref. [15], simulations were performed by Molecular Dynamics (MD), using the PARCAS simulation code [21]. However, in principle any MD code could be used, such as the open-source LAMMPS code [22]. Furthermore, if the time required for MD simulation is a significant obstacle, then the simpler, faster Binary Collision Approximation (BCA) becomes an appealing approach. A variety of BCA codes exist, including many descended from the widely-used SRIM code [23, 24, 25] such as TRIDYN [26, 27], SDTRIMSP [28] and TRI3DST [29]. Importantly, these various codes have different and complementary capabilities, and so practitioners may need to employ different codes to answer different types of questions. To facilitate this, the library should in principle be compatible with as many as possible.

**Framework Summary.** In response to these motivations, we have therefore implemented our library as a layered framework. All of the analysis routines are implemented using an internal, standardized set of data structures for holding the results of simulation output. These structures, themselves, are then hidden behind a set of commonly-used high-level routines for generating coefficient values, which may be easily used by practitioners without reproducing any of the underlying work. Furthermore, the framework is designed to be agnostic with respect to the simulation tool used to obtain the impact data. Any solver capable of producing such data can be “wrapped” within an input-output object whose job is simply to write the input files required by that solver, run the simulation, and read the resulting output files. Because input and output file formats vary widely between codes, the specifics of each wrapper are abstracted from the user.

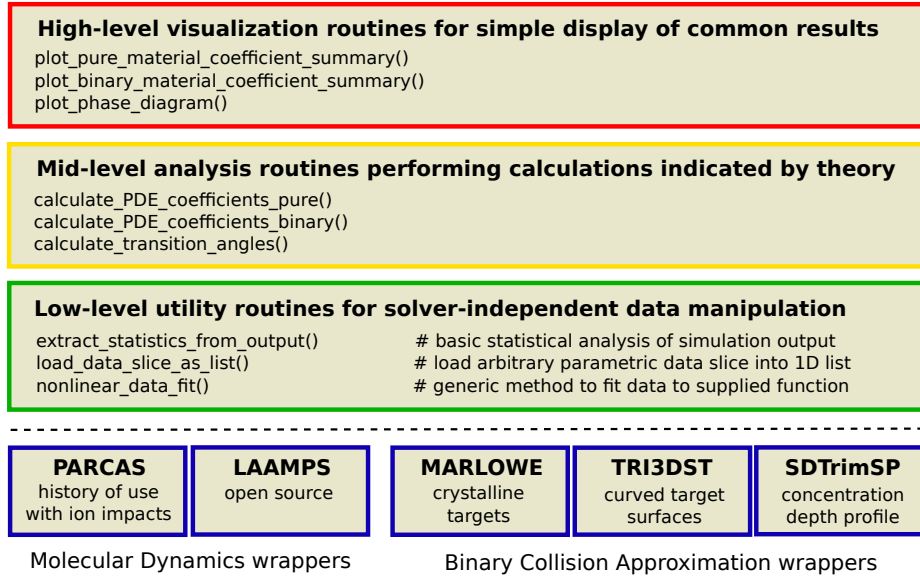


Figure 1: A schematic diagram of the organization of the PyCraters library. (a) At the lowest level are solver-specific wrappers that abstract the performance of individual simulations behind a standardized interface. Requests for simulations are performed, and results obtained, in a standard format. (b) Above this layer is a set of generic utilities for the collection of statistics from simulation data, the storage and loading of data in easy-to-use formats, and the fitting of data points to smooth curves. (c) Next is a layer for performing the real work associated with the crater function formalism – the loading of appropriate slices of data, fitting that data to appropriate curves, and differentiating the resulting fits to obtain coefficients of linearized equations. (d) Finally, a related set of visualization utilities plots commonly-sought coarse-grained quantities. On top of this framework, small end-user codes to perform basic surveys can easily be written by non-specialists, while specialists can directly invoke the lower-level routines to suit their particular needs.

Instead, simulation parameters common to all solvers are available in a simplified, standardized, high-level user interface common to all wrappers, whereas features specific only to certain solvers can be specified on a solver-by-solver basis.

The overall approach is illustrated in Figure 1, and consists of components at various levels of abstraction. From lowest-level to highest-level, these may be summarized as consisting of

1. Wrappers around individual Molecular Dynamics or Binary Collision Approximation solvers.
2. Generic, but extensible analysis routines for the extraction of moments and other statistics.
3. An customizable library for the fitting of these statistics to appropriate smooth functional forms.
4. Routines for performing calculations needed to convert fitted functions into PDE coefficients.
5. Plotting utilities to display reasonable summaries of various kinds of data.

It should be stressed, however, that from the user’s perspective, these capabilities are visible in the opposite order. For example, a user desiring to plot the PDE coefficients for a given material would simply call an associated plotting function. This function checks to see if fits are available from a prior use of the script, and if not, requests such fits from a lower-level function. That function checks to see whether the statistical data are available, and if not, requests them from a still lower-level function. Finally, that function invokes a BCA or MD simulation tool wrapper, which performs all communication with the solver needed to obtain the statistics.

---

**Algorithm 1** Common code needed to load the PyCraters libraries.

---

```
1 # import necessary libraries
2 import sys
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import pycraters.wrappers.TRI3DST as wrap
6 import pycraters.helpers as help
7 import pycraters.IO as io
8
9 # build solver wrapper and parameter object
10 exec_location = sys.argv[1]
11 wrapper = wrap.TRI3DST_Wrapper(exec_location)
12 params = wrap.TRI3DST_Parameters()
```

---

We conclude this section by noting that our goal is not to completely wrap the functionality of sophisticated tools such as LAAMPS and TRIDYN – rather, it is to provide a consistent interface to these tools for users with the specific aim of using them to gather and apply single-impact crater function statistics. Python – a high-level scripting language with very mature numerical capabilities – is an ideal language in which to implement such an interface. We note that although certain kinds of parameter sweeping are built into some of the solvers we have discussed, such capabilities effectively represent miniature “scripting languages” unique to each solver. By using a mature language like Python to perform all scripting, the PyCraters library allows parameter sweeping and statistical analysis to be done in a uniform manner, regardless of the underlying tool used to perform the simulations, and therefore without having to learn the details of the scripting capabilities of each solver.

## 4 Usage Examples

We will now proceed to briefly illustrate several examples of the framework in use. The focus of this section is not on any particular result obtained herein (which are subject to revision as theoretical approaches improve), nor on precisely documenting the codebase (which is subject to change in future software versions). Rather, it is to emphasize the general structure of the code as depicted in Figure 1, and illustrate the kinds of problems that the framework has been designed to investigate. In all cases, we will estimate PDE coefficients using the simpler Eqs. (5), procedures for which have been documented in the literature. A report on somewhat more complicated procedures for using the revised Eqs. (7) will be the topic of future work.

### 4.1 Preliminaries: Basic library loading and setup

We begin with a very brief introduction to the code typically needed at the beginning of a PyCraters script, shown in Algorithm 1. It shows the loading of common mathematical libraries, as well as a wrapper and set of helper routines from the PyCraters library. The first code block simply loads all needed libraries, including the relevant parts of PyCraters itself. The second code block reads the executable location from the command line, and creates the two main objects needed to interact with the library – a solver wrapper and a parameter holder. The latter is tasked with describing the simulation environment, while the former abstracts each solver behind a uniform interface. Note that in these two code blocks, the user chooses the TRI3DST solver [29]. If a different solver were desired, only three lines of code would need to be changed.

### 4.2 Simplest example: Angle-dependence over one energy

We now present a very simple illustration of the framework, by using it to obtain results of the kind reported in Ref [15] – PDE coefficients associated with the low-energy irradiation of a pure material. For narrative consistency with the next section, we choose  $100\text{ eV Ne}^+ \rightarrow \text{C}$ . The code needed for this example appears

---

**Algorithm 2** An example program listing using the PyCraters Python framework.

---

```
1 # set parameter values in high-level format
2 params.target = ["C", 1.0]
3 params.beam   = "Ne"
4 params.energy = 100
5 params.angle  = None
6 params.impacts = 1000
7
8 # perform simulations over a series of angles
9 angles = np.linspace(0,85,18)
10 for aa in angles:
11     params.angle = aa
12     wrapper.go(params)
13
14 # extract statistics, perform fits, and generate plots
15 fitdata = help.extract_PDE_coefficients(params, angles)
16 results = help.plot_coefficient_summary(fitdata)
17 plt.show()
```

---

in Algorithm 2, and should be assumed to be preceded by the code in Algorithm 1. The first code block specifies the environmental parameters the user wishes to consider, including the ion and target species, the ion energy and incidence angle, and the number of impacts to perform (the incidence angle is left blank). The second code block sweeps over the incidence angle in 5-degree increments: for each angle, it updates the parameter holder, and calls the solver wrapper for the specified set of parameters. Finally, the third code block calls a routine that plots a simple summary of the results.

The bulk of the work performed by the PyCraters framework is hidden behind just three function calls. The call to the `wrapper.go()` routine performs all interaction with the underlying BCA solver, including writing input files, calling the executable, reading output files, extracting moments, and storing the results on the hard disk under a unique file name constructed by the `Params()` object. Next, the call to the `help.extract_PDE_coefficients()` routine reads these files for all angles in the sweep, uses a self-contained library to fit each of the angle-dependent moments to appropriate functions of the incidence angle, performs differentiations needed to construct the coefficients, and stores both fits and coefficients under additional reconstructable filenames. Finally, the `help.plot_coefficient_summary()` routine is a relatively simple visualization routine containing plots likely to be of interest to a casual user. For the program listing in Algorithm 2, the output is exhibited in Fig. 2.

### 4.3 Angle-Energy Phase Diagrams

In this section we present a slightly more complicated example, demonstrating the ease with which additional parameters may be swept to identify trends in statistical behavior. Specifically, we observe that because the stability of Eqn. 4 is determined by the signs of the angle-dependent coefficients  $S_X(\theta)$  and  $S_Y(\theta)$ , the sixth panel of Figure 2 describes the transition in expected behavior from flat surfaces (both coefficients positive) to ripples oriented in the  $x$ -direction ( $S_X < 0$ ). The points at which these curves cross the origin, and each other, thus serve to divide domains of different expected behavior. Using the PyCraters framework, it is only a matter of adding an extra `for()` loop to repeat this calculation for a variety of ion energies, and thereby obtain an angle-energy phase diagram. Furthermore, additional sweeps over ion and target species allow the associated phase diagrams for each ion/target combination to be compared, enabling the identification of trends in stability with respect to ion and target atom mass. The code used to perform these simulation is listed as Algorithm 3. We note the great similarity to the code listed in Algorithm 2 – with the exception of three extra nested `for()` loops, and the additional calls to functions like `help.find_pattern_transitions()` (which calculates curve intersections between  $\{0, S_X(\theta), S_Y(\theta)\}$ ) and `help.plot_energy_angle_phase_diagram()` (which plots the results) – the majority of the code is similar.

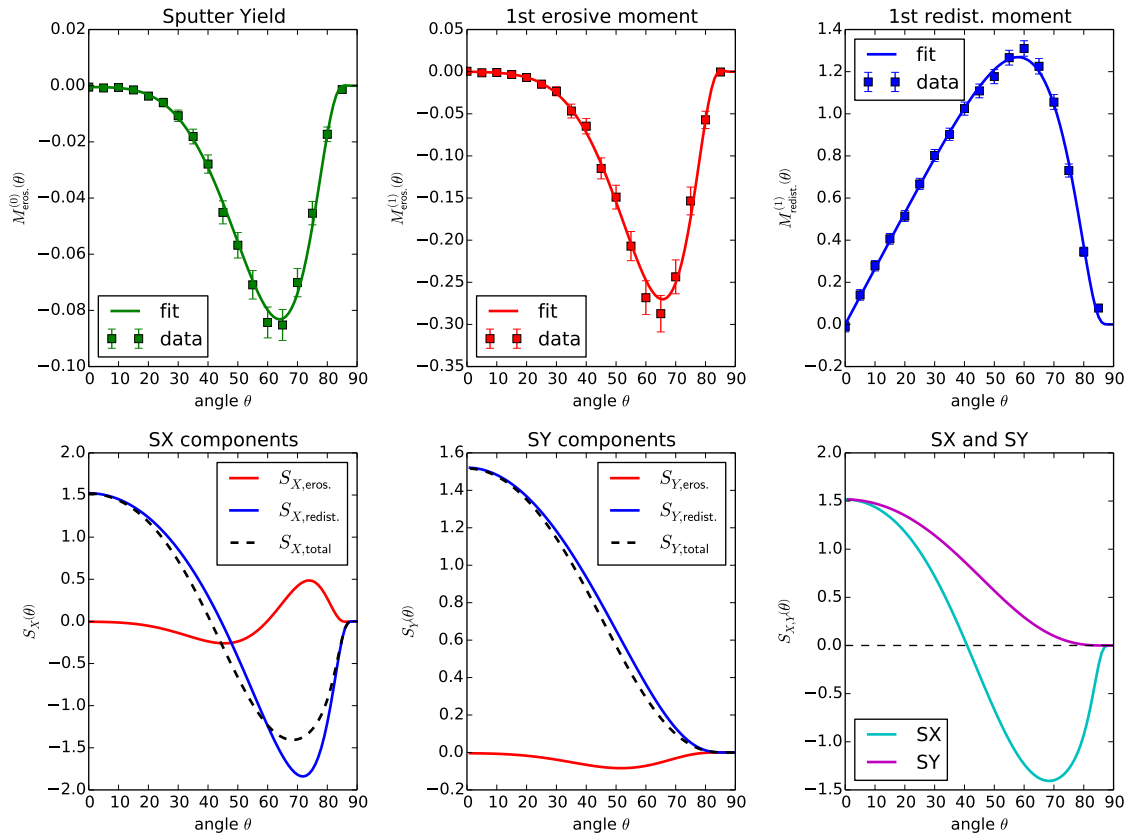


Figure 2: Output of the program listing in.



---

**Algorithm 3** A program for generating angle-energy phase diagrams over many ion/target combination.

---

```

1 # identify parameters to sweep over
2 targets = [ ["C", 1.0], ["Si", 1.0], ["Ge", 1.0], ["Sn", 1.0]]
3 beams   = ["Ne", "Ar", "Kr", "Xe"]
4 energies = 10.*(np.linspace(2.0, 4.0, 21))
5 angles   = np.linspace(0,85,18)
6 findeg   = np.linspace(0, 90, 91)
7 impacts  = 1000
8
9 # perform the simulations and store the results
10 for tt in targets:
11     for bb in beams:
12         tdatalist = []
13         for ee in energies:
14             for aa in angles:
15                 params.target = tt
16                 params.beam   = bb
17                 params.energy = ee
18                 params.angle  = aa
19                 params.impacts = impacts
20                 wrapper.go(params)
21
22             # extract coefficients and find transition angles
23             fitdata = help.extract_PDE_coefficients(params, angles)
24             tdata    = help.find_pattern_transitions(fitdata)
25             tdatalist.append(tdata)
26
27         # after each energy sweep, plot and save the phase diagram
28         help.plot_energy_angle_phase_diagram(tdatalist)
29         plt.savefig("%s-%s-phase-diagram.svg" % (ion_species, target_species))

```

---

This reflects the aims of the framework of providing high-level functionality in easy-to-use functions, allowing the end user to focus on specifying the range of parameters to be explored.

A sampling of the respective graphs are shown in Figure (3). In the first column, the ion mass is increased, and a corresponding increase in the size of the region for stable, flat surfaces is observed, as well as a decrease in the size of the region for perpendicular mode ripples. Because the stable regions are induced by redistribution, and the latter by erosion, this indicates an increasing relative strength of the redistributive effect as the ion mass increases. By contrast, in the second column, the target mass is increased, and the trend is reversed – the stable region shrinks, and the region of perpendicular mode ripples grows. This indicates that generically, as the target mass increases, the role of erosion grows relative to that of mass redistribution. Interestingly, for most ion/mass combinations, the effect of the ion energy seems to be small above around 500 eV. It should be stressed, however, that these results are not presented as a definitive prediction on behavior. They capture only the effect of the collision cascade – sputter erosion and mass redistribution – and do not capture phenomena such as stress implantation and relaxation [30, 31, 32, 33]. This effect also contributes to the coefficients  $S_X(\theta)$  and  $S_Y(\theta)$ , and its magnitude relative to collisional effects is not yet known.

## 4.4 Binary Materials

The simulation and analysis of single-impact events on binary targets is easily performed within the PyCraters framework. In Ref. [16], the Crater Function approach was extended to binary compound targets, and estimates were made of several coefficients within a theory describing the irradiation of such targets [12, 34],

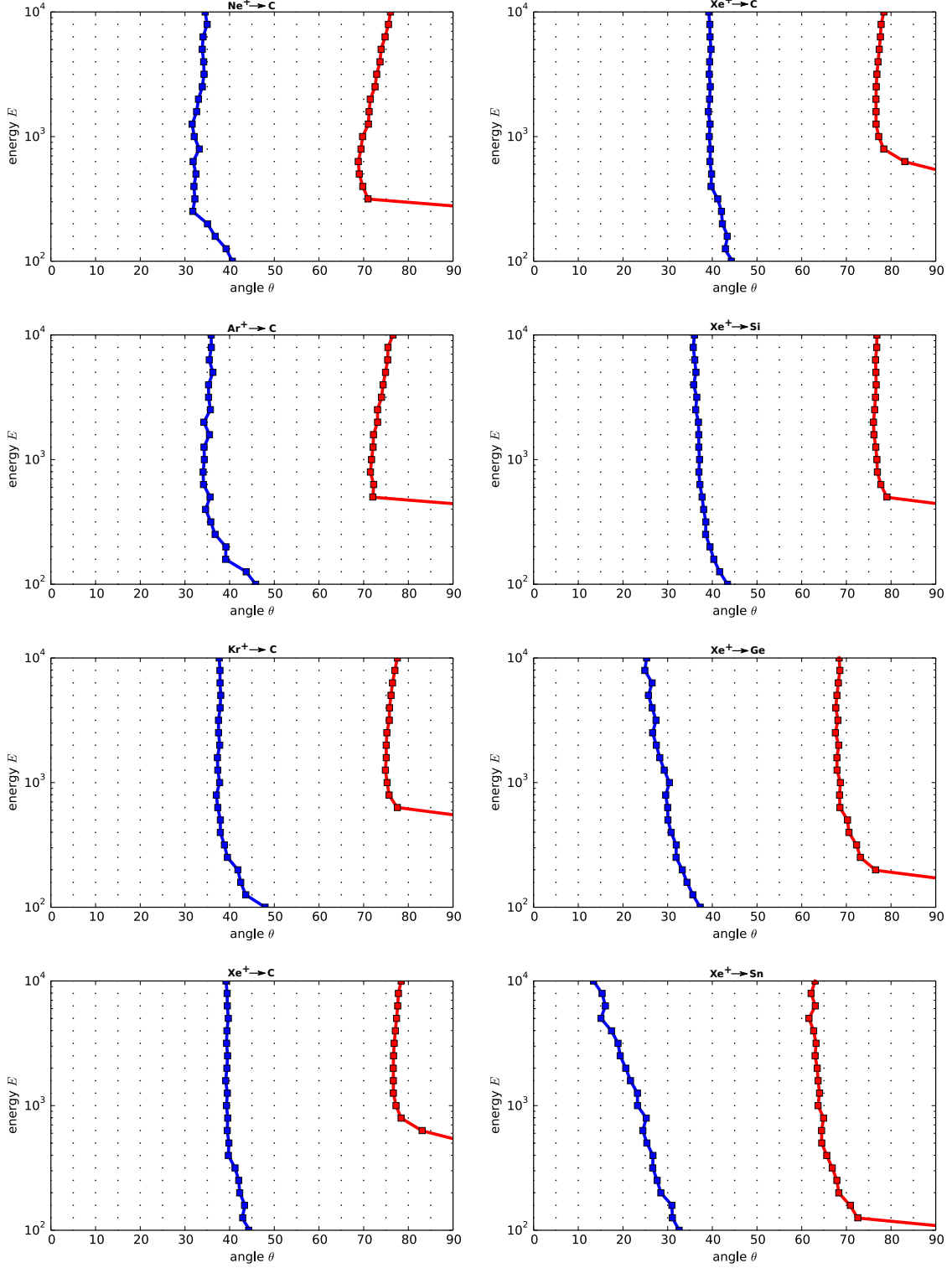


Figure 3: A collection of angle-energy phase diagrams generated using the program listed in Algorithm 3. Left column descending: increasing ion mass ( $\{\text{Ne}^+, \text{Ar}^+, \text{Kr}^+, \text{Xe}^+\} \rightarrow \text{C}$ ). Right column descending: increasing target mass ( $\text{Xe}^+ \rightarrow \{\text{C}, \text{Si}, \text{Ge}, \text{Sn}\}$ ). In each figure, the region between the left edge and the blue line indicates flat targets, the region between the blue and red lines indicates parallel mode ripples, and the region between the red line and the right edge indicates perpendicular mode ripples.

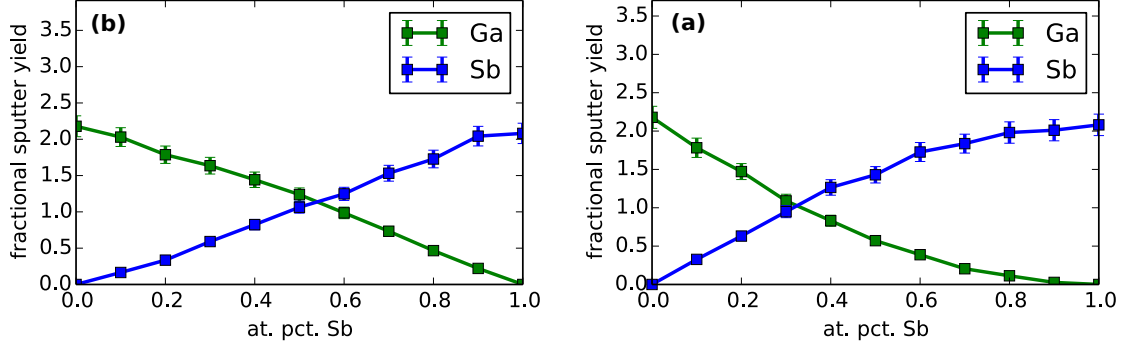


Figure 4: Estimates of the relative sputter yields for various amorphous compounds  $\text{Ga}_{1-x}\text{Sb}_x$  (a) if the target is homogeneous, (b) if the target exhibits a 1nm layer enriched in Sb of the form  $\text{Ga}_{1-y}\text{Sb}_y$ , with  $y = 2x - x^2$ . The enrichment of the surface layer produces a dramatic shift in the equilibrium film concentration.

for the case of GaSb irradiated by Ar at 250 eV. However, during this process, a notable discrepancy between simulations and experiments was observed – whereas experiments observe gradual buildup of excess Ga on the irradiated target, the simulations of impacts on GaSb indicated a slight preferential sputtering of Ga, which would lead to excess Sb. It was hypothesized that this discrepancy might be due to the effect of Gibbsian surface segregation, whereby the atom with the lower surface free energy (in this case Sb) migrates to the surface [35], where it is more easily sputtered due to surface proximity. Because the in-situ composition profile is unknown, the targets used in the initial studies were homogeneous, and could not capture this effect.

The PyCraters framework is an ideal tool to explore this hypothesis, and a code listing which performs the needed simulations is provided in Algorithm 4. In this example, the user must move beyond built-in functionality, and begin to provide some customization to the default settings. For instance, we specify a list of amorphous targets with varying stoichiometries of the form  $\text{Ga}_{1-x}\text{Sb}_x$ , and a simple user-supplied function that allows a surface layer 1nm thick to be modified by enriching it in Sb from  $x$  to a plausible level of  $y = 2x - x^2$ . After the target is then irradiated at 1 keV by Ar+, and the relative sputter yields of Ga and Sb (which appear in the zeroth moment of the crater function) are extracted directly from the storage files, and a few lines of code are written to plot these yields as functions of the base Sb composition.

The results for both unmodified and modified targets are shown in Figure 4. Unsurprisingly, the modified target exhibits a greater sputter yield of Sb than the unmodified target – the top monolayer of this target contains more Antimony, and most sputtering occurs from the top monolayer. In fact, the new yields largely mirror the enriched layer composition. Interestingly, however, this relatively small modification is entirely sufficient to resolve the discrepancy between simulation and experiment just described. Though the actual composition profile remains as yet unknown, the results of this kind of experimentation with plausible model targets suggests that this mechanism is likely sufficient to explain the observed enrichment of Ga over time.

#### 4.5 Future work: Comparison of Methodologies

An important future goal of the framework is to facilitate the comparison of simulations performed using Molecular Dynamics to those using the Binary Collision Approximation. For instance, using MD, Ref. [15] found redistributed atoms to contribute far more to the shape and magnitude of coefficients in Eqs. (5) than did sputtered atoms. However, using the BCA, Ref. [36] reports a reduced redistributive signal for the environment studied in Ref. [15], and finds that sputtered atoms are dominant for most angles at higher energies. The use of MD vs BCA may be an important source of these conflicting results – in Ref. [37], it was found that the BCA reports significantly fewer displacements of small magnitude than molecular dynamics. Because of the very large number of such displacements, they may contribute significantly to the effect of mass redistribution, and hence the BCA approach may systematically under-report the strength of this effect. This question demands further study, and because the extraction of the statistics and the

---

**Algorithm 4** Code to investigate the effect of segregation.

---

```
1 # define a simple function of depth
2 def concentration_function(params, depth):
3     cbase = [ a[1] for a in params.target ]
4     if depth > 10:
5         return cbase
6     else:
7         bSb = cbase[1]
8         lSb = 2.0*bSb - bSb**2
9         return [1-lSb, lSb]
10
11 # basic parameter setup
12 params.target = None
13 params.beam = "Ar"
14 params.angle = 0.0
15 params.energy = 1000
16 params.impacts = 1000
17 params.set_parameter("cfunc", concentration_function)
18
19 # set up targets at different concentrations
20 targets = [ ]
21 for phi in np.linspace(0.0, 1.0, 11):
22     target = ["Ga", 1.0-phi], ["Sb", phi]
23     targets.append(target)
24
25 # iterate over targets
26 for tt in targets:
27     params.target = tt
28     wrapper.go(params)
29
30 # make plots
31 import libcraters.IO as io
32 m0e_avg = io.array_range( './', params, 'target', targets, 'm0e_avg' )
33 m0e_std = io.array_range( './', params, 'target', targets, 'm0e_std' )
34 Gaval = [ a[0] for a in m0e_avg ]
35 Sbval = [ a[1] for a in m0e_avg ]
36 Gaerr = [ b[0]/np.sqrt(params.impacts) * 1.97 for b in m0e_std ]
37 Sberr = [ b[1]/np.sqrt(params.impacts) * 1.97 for b in m0e_std ]
38
39 plt.figure(1, figsize=(4.5, 3.0))
40 pctSb_list = [ a[1][1] for a in targets ]
41 plt.errorbar(pctSb_list, -np.array(Gaval), yerr=Gaerr, label='Ga', fmt='gs-', linewidth=2)
42 plt.errorbar(pctSb_list, -np.array(Sbval), yerr=Sberr, label='Sb', fmt='bs-', linewidth=2)
43 plt.xlabel('at. pct. Sb')
44 plt.ylabel('fractional sputter yield')
45 plt.legend() plt.ylim(0, max(-np.array(Tvals))*1.7 )
46 plt.tight_layout() plt.show()
47 plt.savefig('relative-sputter-yields.svg')
```

---

estimation of coefficients are generic processes independent of the solver within the PyCraters library, it will be straightforward to use those common resources to identify the implications of these observations for the specific statistics required by crater function analysis.

In a related vein, several different procedures for extracting the crater function  $\Delta h$  and its moments  $M^{(i)}$  from a list of atomic positions have been suggested in the literature [38, 15, 39, 36], and the importance of the differences between the strategies has never been investigated. The PyCraters library provides a natural environment in which to conduct such studies. Because the framework provides the results of simulations in a common format, variations on the statistical extraction routine can be written in a way that is independent of both the underlying solver *and* the methods used to fit and differentiate the results. This should allow easy comparison of the different extraction methods for a variety of simulation environments, and aid the identification of best practices for this important procedure.

## 5 Summary

In conclusion, we have introduced a Python framework designed to automate the most common tasks associated with the extraction and upscaling of the statistics of single-impact crater functions to inform coefficients of continuum equations describing surface morphology evolution. The framework has been designed to be compatible with a wide variety of existing atomistic solvers, including Molecular Dynamics and Binary Collision Approximation codes. However, in order to remain accessible to first-time users, the details of each of these solvers is abstracted behind a standardized interface, and much functionality can be accessed via high-level functions and visualization routines. Although the addition of much functionality is still in progress, the current codebase is able to reproduce many important results from the recent literature, and examples demonstrating these capabilities are included to facilitate modification and additional exploration by the community. The project is currently hosted on the BitBucket repository under a suitable open-source license, and is available for immediate download.

## References

- [1] M. Navez, D. Chaperot, , and C. Sella. Microscopie electronique - etude de l'attaque du verre par bombardement ionique. *Comptes Rendus Hebdomadaires Des Seances De L Academie Des Sciences*, 254:240, 1962.
- [2] M. J. Baldwin and R. P. Doerner. Helium induced nanoscopic morphology on tungsten under fusion relevant plasma conditions. *Nucl. Fusion*, 48:035001, 2008.
- [3] S. Facsko, T. Dekorsy, C. Koerdt, C. Trappe, H. Kurz, A. Vogt, and H. L. Hartnagel. Formation of ordered nanoscale semiconductor dots by ion sputtering. *Science*, 285:1551–1553, 1999.
- [4] P. Sigmund. Theory of sputtering. I. Sputtering yield of amorphous and polycrystalline targets. *Phys. Rev.*, 184:383–416, 1969.
- [5] P. Sigmund. A mechanism of surface micro-roughening by ion bombardment. *J. Mater. Sci.*, 8:1545–1553, 1973.
- [6] R. M. Bradley and J. M.E. Harper. Theory of ripple topography induced by ion bombardment. *J. Vac. Sci. Technol.*, 6:2390–2395, 1988.
- [7] G. Carter and V. Vishnyakov. Roughening and ripple instabilities on ion-bombarded si. *Phys. Rev. B*, 54:17647–17653, 1996.
- [8] B. P. Davidovitch, M. J. Aziz, and M. P. Brenner. On the stabilization of ion sputtered surfaces. *Phys. Rev. B*, 76:205420, 2007.
- [9] G. Ozaydin-Ince and K. F. Ludwig Jr. In situ x-ray studies of native and mo-seeded surface nanostructuring during ion bombardment of si(100). *J. Phys. Cond. Matt.*, 21:224008, 2009.
- [10] S. Macko, F. Frost, B. Ziberi, D.F. Forster, and T. Michely. Is keV ion-induced pattern formation on Si(001) caused by metal impurities? *Nanotechnology*, 21:085301, 2010.
- [11] V. B. Shenoy, W. L. Chan, and E. Chason. Compositionally modulated ripples induced by sputtering of alloy surfaces. *Physical Review Letters*, 98:256101, 2007.
- [12] R. Mark Bradley and Patrick D. Shipman. Spontaneous pattern formation induced by ion bombardment of binary compounds. *Physical Review Letters*, 105:145501, 2010.
- [13] S. A. Norris. Ion-assisted phase separation in compound films: An alternate route to ordered nanostructures. *Journal of Applied Physics*, 114:204303, 2013.
- [14] S. A. Norris, M. P. Brenner, and M. J. Aziz. From crater functions to partial differential equations: A new approach to ion bombardment induced nonequilibrium pattern formation. *J. Phys. Cond. Matt.*, 21:224017, 2009.
- [15] S. A. Norris, J. Samela, L. Bukonte, M. Backman, D. F. K. Nordlund, C.S. Madi, M.P. Brenner, and M.J. Aziz. Molecular dynamics of single-particle impacts predicts phase diagrams for large scale pattern formation. *Nature Communications*, 2:276, 2011.
- [16] Scott A. Norris, J. Samela, K. Nordlund, M. Vestberg, and M. Aziz. Crater functions for compound materials: a route to parameter estimation in coupled-pde models of ion bombardment. *Nuclear Instruments and Methods in Physics Research B*, 318B:245–252, 2014. arXiv:1303.2674.
- [17] Matt P. Harrison and R. Mark Bradley. Crater function approach to ion-induced nanoscale pattern formation: Craters for flat surfaces are insufficient. *Physical Review B*, 89:245401, 2014.
- [18] M. J. Aziz. Nanoscale morphology control using ion beams. *Matematisk-fysiske Meddelelser*, 52:187–206, 2006.

- [19] E. M. Bringa, K. Nordlund, and J. Keinonen. Cratering energy regimes: From linear collision cascades to heat spikes to macroscopic impacts. *Phys. Rev. B*, 64:235426, 2001.
- [20] G. Costantini, F. Buatier de Mongeot, C. Boragno, and U. Valbusa. Is ion sputtering always a “negative homoepitaxial deposition”? *Phys. Rev. Lett.*, 86:838–841, 2001.
- [21] K. Nordlund. PARCAS computer code. The main principles of the molecular dynamics algorithms are presented in [40, 41]. The adaptive time step and electronic stopping algorithms are the same as in [42].
- [22] Open-source LAMMPS molecular-dynamics code hosted at <http://lammps.sandia.gov/>. The original algorithms of this community-maintained codebase are presented in Ref. [43].
- [23] J. P. Biersack and L. G. Haggmark. A monte carlo computer program for the transport of energetic ions in amorphous targets. *Nuclear Instruments and Methods*, 174:257–269, 1980.
- [24] J. F. Ziegler, J. P. Biersack, and U. Littmark. *The Stopping and Range of Ions in Matter*. Pergamon Press, New York, 1985.
- [25] SRIM, 2000.
- [26] W. Möller and W. Eckstein. TRIDYN – a TRIM simulation code including dynamic composition changes. *Nucl. Inst. Meth. Phys. Res. B*, 2:814–818, 1984.
- [27] W. Möller. TRIDYN-HZDR.
- [28] A.Mutzke, R.Schneider, W.Eckstein, and R.Dohmen. SDTrimSP version 5.0. IPP Report 12/8, Garching, 2011.
- [29] M. L. Nettiadi, L. Sandoval, H. M. Urbassek, and W. Möller. Sputtering of Si nanospheres. *Physical Review B*, 90:045417, 2014.
- [30] Mario Castro and Rodolfo Cuerno. Hydrodynamic approach to surface pattern formation by ion beams. *Applied Surface Science*, 258:4171–4178, 2012.
- [31] S. A. Norris. Stability analysis of a viscoelastic model for ion-irradiated silicon. *Physical Review B*, 85:155325, 2012.
- [32] S. A. Norris. Stress-induced patterns in ion-irradiated silicon: model based on anisotropic plastic flow. *Phys. Rev. B*, 86:235405, 2012. arxiv:1207.5754.
- [33] M. Castro, R. Gago, L. Vázquez, J. Muñoz-García, and R. Cuerno. Stress-induced solid flow drives surface nanopatterning of silicon by ion-beam irradiation. *Physical Review B*, 86:214107, 2012.
- [34] P. D. Shipman and R. M. Bradley. Theory of nanoscale pattern formation induced by normal-incidence ion bombardment of binary compounds. *Physical Review B*, 84:085420, 2011.
- [35] W. Yu, J. L. Sullivan, and Saied S. O. Xps and leiss studies of ion bombarded gasb, insb and cdse surfaces. *Surface Science*, 352–354:781–787, 1996.
- [36] Hans Hofsäss. Surface instability and pattern formation by ion-induced erosion and mass redistribution. *Applied Physics A*, 114:401–422, 2014.
- [37] L. Bukonte, F. Djurabekova, J. Samela, K. Nordlund, S.A. Norris, and M.J. Aziz. Comparison of molecular dynamics and binary collision approximation simulations for atom displacement analysis. *Nuclear Instruments and Methods in Physics Research B*, 297:23–28, 2013.
- [38] N. Kalyanasundaram, M. Ghazisaeidi, J. B. Freund, and H. T. Johnson. Single impact crater functions for ion bombardment of silicon. *Appl. Phys. Lett.*, 92:131909, 2008.
- [39] M. Z. Hossain, K. Das, J. B. Freund, and H. T. Johnson. Ion impact crater asymmetry determines surface ripple orientation. *Applied Physics Letters*, 99:151913, 2011.

- [40] K. Nordlund, M. Ghaly, R. S. Averback, M. Caturla, T. Diaz de la Rubia, and J. Tarus. Defect production in collision cascades in elemental semiconductors and fcc metals. *Phys. Rev. B*, 57:7556–7570, 1998.
- [41] M. Ghaly, K. Nordlund, and R. S. Averback. Molecular dynamics investigations of surface damage produced by kiloelectronvolt self-bombardment of solids. *Philos. Mag. A*, 79:795–820, 1999.
- [42] K. Nordlund. Molecular dynamics simulation of ion ranges in the 1 – 100 kev energy range. *Comput. Mater. Sci.*, 3:448, 1995.
- [43] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117:1–19, 1995.